
This is the **published version** of the bachelor thesis:

Osuna Jeronimo, Òscar. *Disseny i implementació d'un SaaS per a la gestió de dispositius de xarxa FortiGate*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334462>

under the terms of the  license.

Disseny i implementació d'un SaaS per a la gestió de dispositius de xarxa FortiGate

Òscar Osuna Jerónimo

Resum—La gestió centralitzada de dispositius de firewall FortiGate sol dependre de programari empresarial complex i costós, fora de l'abast de moltes petites empreses. Davant d'aquesta barrera, aquest treball presenta Gorget, una plataforma SaaS per a la monitorització i gestió de dispositius de firewalls FortiGate. El sistema permet als usuaris registrar-se de manera segura, organitzar els seus dispositius en organitzacions aïllades i supervisar-ne l'estat i el rendiment de manera autònoma i contínua. Mitjançant un procés de polling periòdic i concurrent, l'aplicació consulta cada dispositiu, en recull les mètriques (ús de CPU, memòria, disc i sessions actives) i les emmagatzema en una base de dades de sèries temporals per mostrar-ne l'evolució mitjançant gràfics. El projecte s'ha construït sobre una arquitectura hexagonal, s'ha validat amb un dispositiu FortiGate físic real connectat a través d'internet, i s'ha desplegat en un entorn de producció amb domini propi i comunicació xifrada. Més enllà del producte, el treball demostra la viabilitat de construir una eina de gestió de firewalls accessible i de baix cost.

Paraules clau—Monitorització de xarxes, Firewalls, FortiGate, SaaS, API REST, FastAPI, InfluxDB, Polling.

Abstract—Centralized management of FortiGate firewall devices typically relies on complex and expensive enterprise software, out of reach for many small businesses. Faced with this barrier, this work presents Gorget, a SaaS platform for the monitoring and management of FortiGate firewall devices. Gorget is a SaaS platform for the monitoring and management of FortiGate firewall devices. The system allows users to register securely, organize their devices into isolated organizations, and supervise their status and performance autonomously and continuously. Through a periodic and concurrent polling process, the application queries each device, collects its metrics (CPU, memory, disk usage and active sessions) and stores them in a time-series database to display their evolution through charts. The project is built on a hexagonal architecture, has been validated with a real physical FortiGate device connected over the internet, and has been deployed to a production environment with its own domain and encrypted communication. Beyond the product itself, this work demonstrates the feasibility of building an accessible and low-cost firewall management tool.

Index Terms—Network monitoring, Firewalls, FortiGate, SaaS, REST API, FastAPI, InfluxDB, Polling.



1 INTRODUCCIÓ - CONTEXT DEL TREBALL

En l'actualitat, la ciberseguretat i la connectivitat depenen en gran mesura de dispositius de xarxa robustos com els FortiGate. No obstant això, la gestió centralitzada d'aquests dispositius sol requerir coneixements tècnics molt avançats o infraestructures costoses (com FortiManager). Això provoca la delegació d'aquests serveis a empreses externes i la dependència de tècnics especialitzats, que una empresa de baix nivell potser no es pot permetre.

Aquest projecte neix de la necessitat de democratitzar i simplificar aquesta gestió mitjançant un model SaaS (Software as a Service). L'objectiu és oferir una plataforma intuïtiva que permeti a empreses i proveïdors de serveis gestionats (MSPs) monitorar, configurar i mantenir la seva flota de dispositius des d'un sol punt, sota la filosofia de "configurar i oblidar" (set and forget).

2 OBJECTIUS

L'objectiu general del projecte és desenvolupar un Software as a Service (SaaS) per a la monitorització i gestió de dispositius FortiGate, accessible des de qualsevol navegador i desplegat en un entorn de producció real.

Per assolir aquest objectiu general, es defineixen els següents objectius específics:

-
- E-mail de contacte: oscarosunajerónimo@gmail.com
 - Menció realitzada: Enginyeria del Software
 - Treball tutoritzat per: Jorge Bernal del Nozal (Departament de Ciències de la Computació)
 - Curs 2025/26

- Desenvolupar un sistema d'autenticació segur que permeti als usuaris registrar-se i iniciar sessió.
- Implementar un model multi-tenant d'organitzacions, de manera que cada usuari pugui gestionar els seus propis dispositius de manera aïllada.
- Dissenyar i implementar un sistema de monitorització autònom que consulti periòdicament l'estat i les mètriques dels dispositius sense intervenció manual.
- Emmagatzemar l'històric de mètriques en una base de dades de sèries temporals i mostrar-ne l'evolució mitjançant gràfics.
- Desenvolupar la interfície gràfica per visualitzar l'estat dels dispositius i les seves dades de manera senzilla i intuïtiva.
- Connectar el sistema amb un dispositiu FortiGate físic real a través de la seva API.
- Desplegar el sistema en un entorn de producció, amb un domini propi i comunicació xifrada (HTTPS).
- Definir una identitat de marca per al producte (nom, colors i visió).

3 REQUISITS HARDWARE/SOFTWARE

Per al desenvolupament i desplegament del projecte es defineixen una sèrie de requisits, tant de maquinari com de programari.

3.1 Hardware

L'equip de desenvolupament és l'ordinador on s'escriu i es prova el codi, i ha de poder executar simultàniament l'IDE, el servidor de backend, el client web i les eines de desenvolupament. L'equip emprat ha estat un MacBook Pro de 16 polzades del 2019, amb un processador Intel Core i7-9750H de sis nuclis a 2,6 GHz, 16 GB de memòria RAM i el sistema operatiu macOS 15.6.1.

El servidor de producció allotja el backend i les dues bases de dades. Com s'ha demostrat a l'anàlisi de rendiment (secció 13), un servidor de recursos modestos és suficient. El servidor emprat ha estat un VPS del proveïdor Hetzner [1], situat a Nuremberg (Alemanya), amb 2 vCPU, 4 GB de memòria RAM, 40 GB de disc i sistema operatiu Ubuntu 24.04 LTS.

Finalment, per validar la connexió amb maquinari real cal un dispositiu de proves. En aquest cas s'ha emprat un dispositiu FortiGate model 40F.

3.2 Software

Pel que fa al programari, el desenvolupament s'ha dut a terme amb un IDE com a entorn de treball per escriure el codi, PyCharm. El backend s'ha construït amb el framework FastAPI [2], mentre que el client web s'ha desenvolupat amb SvelteKit [3].

Quant a l'emmagatzematge de dades, s'utilitzen dues tecnologies: PostgreSQL [4], una base de dades relacional per a la informació d'usuaris, organitzacions i dispositius,

i InfluxDB [5], una base de dades de sèries temporals per a l'històric de mètriques. El control de versions del codi s'ha gestionat amb Git [6] i la plataforma GitHub [7].

Finalment, per al desplegament en producció s'han emprat diverses eines de sistema: el servidor web nginx [8], systemd [9] i Let's Encrypt [10].

4 Riscos

Aquest projecte comporta una sèrie de riscos que hem de tenir en compte a l'hora de desenvolupa-lo. Aquests els podem veure a la taula 1:

TAULA 1: Riscos

Risc	Impacte	Contingència
Creixement de les dades de monitorització.	Mig. A llarg termini podria saturar l'emmagatzematge del servidor.	Ús d'una base de dades especialitzada (InfluxDB).
Limitacions de temps per completar el desenvolupament.	Alt. Algunes funcionalitats podrien quedar fora de l'entrega final.	Sistema d'iteracions amb planificació per prioritat.
Seguretat de les dades personals	Alt. Una fuga o un accés no autoritzat exposaria informació confidencial.	Xifrat de contrasenyes amb hash.
No complir la proposta de facilitat d'ús.	Mig. Si la interfície resulta complexa, es perd la proposta de valor.	Disseny d'una interfície senzilla i intuïtiva, i proves d'usuari contínues.

5 METODOLOGIA

El projecte seguirà la metodologia de desenvolupament iterativa incremental àgil en la seva versió Scrum. Seguirem un sistema de iteracions del producte o "èpiques", on cada cop ens aniran apropant al MVP final que presentarem.

Aquest sistema de iteracions està estratègicament pensat per anar construint el projecte des de la base on al final de cada iteració tinguem una versió del producte usable, amb les millores en funcionalitat aplicades. Això ens permet que si no arribem a les últimes iteracions poguem encara així disposar d'una versió usable del projecte, la qual podrem anar millorant en un futur.

Pel que respecta a la qualitat del codi, per cada nou model o funcionalitat al Backend implementat, afegirem una suite de tests per assegurar la integració contínua del

projecte, sense trencar res en el procés. En quant al front, aproximarem una solució més enfocada al testing exploratori, on ens assegurarem que tot continua mostrant-se i comportant-se com s'espera.

6 ESTAT DE L'ART

En l'actualitat, la gestió de dispositius FortiGate es divideix en tres grans blocs:

1. Solucions Oficials (FortiManager): Molt potents però extremadament complexes, cares i pensades per a grans corporacions amb personal certificat.
2. Plataformes Multi-vendor (IspAPP): Permeten configuració al núvol però són genèriques. En intentar servir per a totes les marques, la interfície és feixuga i no aprofita les funcions específiques de Fortinet.
3. Eines de Monitoratge (Domotz, Auvik): Són excel·lents per saber si un equip està caigut, però no permeten gestionar configuracions ni fer backups de forma nativa.

Existeix un segment de PIMES i petits proveïdors que es troben orfes: FortiManager els queda gran (per preu i complexitat) i les eines genèriques no els donen el control que necessiten sobre el firewall. Per tant, trobem una falta d'una solució pel mercat mitjà: un programari que sigui només per a FortiGate, que no requereixi instal·lar servidors i que prioritzi la facilitat d'ús sobre l'excés de funcions.

Davant d'aquest escenari la nostra proposta se centra a ser la "xarxa de seguretat" de l'administrador de FortiGate, prioritzant la utilitat real sobre l'excés de funcionalitats. Els nostres punts clau de diferenciació es basen en primer lloc en una **especialització nativa** que ens enfoca al 100% en FortiGate, a diferència d'eines genèriques, això ens permet oferir una interfície neta i una integració total amb la seva API.

Així mateix destaquem per la **gestió d'històric i de versions**, posant el focus en la **seguretat** en lloc de ser un simple controlador com IspAPP ja que guardem automàticament l'històric de configuracions per permetre comparacions i restauracions ràpides per correccions en cas de que un canvi surti malament. Tot això s'ofereix a través d'un **SaaS** real, eliminant la necessitat de desplegar màquines virtuals o servidors de backup, de manera que l'usuari només ha d'apuntar el seu dispositiu al núvol i començar a gestionar-lo.

A més comptem amb una **UX senzilla** dissenyada per a l'operativa diària, on qualsevol tasca crítica com VPN, regles, reinicis o backups es pot fer en menys de 3 clics, sent ideal per a administradors amb poc temps. Finalment garantim el **millor preu i accessibilitat** per eliminar la barrera d'entrada mitjançant un model transparent que inclou una versió gratuïta per a diversos dispositius amb funcions de monitoratge i backups bàsics, permetent que els administradors provin la plataforma sense compromís abans d'escalar a subscripcions de pagament per a flotes més grans.

7 PLANIFICACIÓ

El projecte tindrà diverses iteracions que aniran afegint funcionalitats, partint de lo més base a lo específic. La planificació en el temps la podem veure a la figura 8 en l'apèndix.

1. Config. inicial i autenticació, 15/03/2026
2. Organitzacions i navegació, 28/03/2026
3. Inventari de dispositius, 11/04/2026
4. Dashboard i detall, 25/04/2026
5. Monitorització, 09/05/2026
6. Connexió dispositiu real, 23/05/2026
7. Testing i desplegament, 14/06/2026

8 ARQUITECTURA DEL PROJECTE

8.1 Arquitectura client servidor

El sistema segueix una arquitectura client-servidor desacoblada, en què la responsabilitat queda clarament separada en dues parts independents que es comuniquen exclusivament a través d'una API REST.

D'una banda tenim el frontend, que no emmagatzema cap tipus de lògica. El seu objectiu és consumir les dades exposades pel backend i mostrar-les en l'estil decidit.

D'altra banda tenim el backend, compost per una base de codi en el framework FastAPI, el qual orquestra la lògica de dades i les exposa per mitjà d'api REST. La persistència es reparteix entre dos sistemes de bases de dades segons la naturalesa de les dades: PostgreSQL per a les dades relacionals tradicionals (usuaris, organitzacions, dispositius, etc) i InfluxDB per a les dades de sèries temporals, com ara les mètriques recollides dels dispositius.

Podem trobar un diagrama més en detall de l'arquitectura del sistema en la figura 12, en l'apèndix.

8.2 Arquitectura del backend

Per executar el nostre sistema el primer pas ha estat la creació del codi del servidor. Per a la creació d'aquest backend, s'ha decidit fer servir l'arquitectura hexagonal [11] seguint els principis de CLEAN architecture [12], organitzada en *features*. Cada *feature* pot ser els usuaris o dispositius, per exemple. Aquesta decisió permet que l'aplicació escales de manera controlada.

L'arquitectura hexagonal tracta de dividir el codi per capes, on cada capa té la seva responsabilitat i només pot parlar amb les capes més internes. Dins de cada *feature* s'apliquen tres capes diferenciades. Això incrementa en gran mesura l'escalabilitat del codi, ja que els mòduls són fàcilment reemplaçables i si, a posteriori migres de base de dades, només toques el mòdul d'interacció amb aquesta.

8.2.1 Les capes

Dintre de cada *feature* dividim les seves funcions i responsabilitats en tres capes ben definides per l'arquitectura hexagonal.

Domini: Conté les entitats del negoci i les regles de domini pures així com els Value Objects, els objectes immutables per modelar dades recollides al nostre sistema. En el nostre cas els DTOs s'implementen com a esquemes de Pydantic [13], fet que aporta validació automàtica.

Aplicació: Conté els casos d'ús (use cases) que són els punt d'entrada a la aplicació i DTOs, els quals transporten dades entre capes i en el nostre cas contenen els esquemes de pydantic.

Infraestructure: Conté les implementacions amb les que es comuniquen amb el món exterior. Inclou els repositoris, un patró del software que encapsula l'accés a les bases de dades per obtenir i modificar dades sense conèixer els detalls de l'implementació, els clients externs, que es comuniquen amb serveis o dispositius de tercers i els models de les bases de dades, que es defineixen com a SQLAlchemy [14] directament en codi.

A la figura 1 es pot veure un diagrama d'aquesta arquitectura hexagonal. A l'apèndix, en la figura 10 se'n troba una versió més detallada pel que fa a l'estructura del codi.

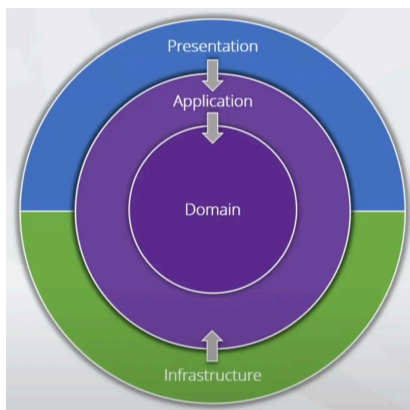


Figura 1: Diagrama de l'arquitectura hexagonal. Font: [24]

8.2 El client web

El client és una aplicació desenvolupada amb SvelteKit. L'aplicació no conté lògica de negoci: la seva única responsabilitat és consumir les dades que exposa l'API i presentar-les. L'enrutament es basa en l'estructura de fitxers: cada carpeta dins de routes/ és una ruta de l'aplicació, de manera que per tenir una pàgina /user/settings només cal afegir-hi un fitxer dins d'una carpeta user. Les pantalles que requereixen estar autenticat s'agrupen sota una mateixa carpeta, separades de les pàgines públiques com el login i el registre.

Tota la comunicació amb el servidor es concentra en un únic mòdul, api.js, que recull el conjunt de funcions que empaqueten les crides a l'API. Aquesta decisió evita repetir la lògica de xarxa per tota l'aplicació i centralitza en un sol lloc detalls com l'adjunció del token d'autenticació a cada petició. Per a l'estat de les variables i dades es distingeixen dos àmbits: les runes gestionen l'estat local d'un component, mentre que els stores permeten compartir estat entre pantalles, com ara l'organització activa.

8.3 Model de dades

8.3.1 Entitats i relacions

L'objectiu del projecte és oferir una plataforma perquè diferents organitzacions puguin controlar i monitoritzar els seus dispositius FortiGate. Per aconseguir-ho, cal modelar dins del sistema totes les entitats que hi intervenen i les relacions que s'estableixen entre elles.

Pel que fa a l'autenticació i el control d'accés, el sistema disposa d'una entitat Usuari. Un usuari pot pertànyer a diverses Organitzacions i una organització pot tenir diversos usuaris. Es tracta doncs, d'una relació de molts a molts. Aquesta relació no és simple: cada vincle entre un usuari i una organització porta associat un rol (per exemple, owner o member), que determina el nivell de permisos.

Una organització pot tenir diversos Grups de dispositius (DeviceGroups), que serveixen per organitzar lògicament els dispositius. Un dispositiu pertany sempre a una organització (relació obligatòria) i opcionalment, pot estar assignat a un grup. D'aquesta manera la jerarquia queda definida així: una organització conté N grups i N dispositius i cada dispositiu pot pertànyer a un grup. En l'apèndix es pot observar la figura 11 amb el diagrama de les entitats i les seves relacions en el sistema.

Amb la mirada en futures ampliacions cap a altres fabricants, els models de dispositiu no estan acoblats de cap manera a FortiGate. El nucli del model és indiferent al fabricant: un camp vendor actua com a discriminador de la plataforma, mentre que els detalls específics de cada fabricant es deleguen a les capes externes. Així si en el futur es volgués donar suport a nous fabricants, només caldria afegir el client que es comunica amb el seu hardware específic, sense modificar l'esquema de dades ni la lògica de negoci.

8.3.2 Models de dades i migracions

L'ORM SQLAlchemy ens permet modelar les bases de dades a través de models en el codi, sense tocar SQL pur. Cada taula es defineix com una classe i cada columna com un atribut tipat, fet que aporta seguretat de tipus i portabilitat.

Per tant de poder actualitzar els models un cop afegim o modifiquem els mateixos tenim Alembic [16]. Aquesta eina ens permet amb una sola comanda crear migracions que modifiquen la base de dades al esquema amb l'estat actual. Cada migració són un fitxer que descriu el canvis en els models de la base de dades.

8.4 El patró API REST

La comunicació entre el client i el servidor segueix l'estil arquitectònic REST [17], un conjunt de convencions per dissenyar APIs sobre HTTP de manera consistent i coherent. En REST cada *feature* és exposada a través d'un endpoint, el qual permet interactuar amb les dades. Discriminem diversos verbs per a diferents accions amb el sistema: GET per obtenir recursos, POST per crear-ne de nous, PUT per modificar i DELETE per eliminar.

Cada resposta inclou un codi HTTP per l'estat de la comanda, 200 significa transacció exitosa i els 400 són pels diversos tipus d'errors. En l'annex podeu trobar tots els endpoints REST exposats pel nostre servidor en la figura 13, ja que FastAPI ens proporciona amb una documentació dinàmica automàticament generada.

8.5 Flux d'una petició: del client a la base de dades

Per il·lustrar com col·laboren totes les capes, aquest és el recorregut complet d'una petició de creació (POST), des que l'usuari interactua amb el client fins que la dada es persisteix:

1. Recepció i validació al router: FastAPI/Pydantic validen el format del Request. En cas negatiu retorna error.
2. Injecció de dependències: Es proveeix una sessió de base de dades activa i usuari autenticat, en cas de ser una ruta protegida.
3. Construcció del cas d'ús: S'instancia el cas d'ús amb el seu repositori.
4. Execució del cas d'ús: S'aplica la lògica de negoci, es retornen errors específics de la mateixa en cas de fallida.
5. Accés a la base de dades: El repositori (infraestructura) executa el SQL via SQLAlchemy.
6. Resposta al client: Serialització automàtica a JSON o conversió d'excepcions a HTTPException.
7. Tractament al frontend: Captura d'errors en un bloc try/catch i actualització de la interfície.

9 DESENVOLUPAMENT

Un cop definides l'arquitectura i les tecnologies es va començar el desenvolupament del sistema. Aquesta secció descriu la implementació de les pantalles principals.

9.1 Inici de sessió i registre

La primera pantalla que troba qualsevol persona que obre Gorget és la d'inici de sessió. És la porta d'entrada al sistema, des d'aquí es pot iniciar sessió amb un compte existent o bé anar a la pantalla de registre per crear-ne un de nou, tal com podem veure a la figura 9.

9.1.1 Sistema d'autenticació i seguretat

Les contrasenyes dels usuaris mai s'emmagatzemen en text pla. En el moment del registre, la contrasenya es transforma mitjançant un algoritme de hash criptogràfic, anomenat bcrypt [18]. Aquest algorisme és unidireccional, per tant no es pot deduir la contrasenya pel seu hash. D'aquesta manera encara que algú trobés el hash de la contrasenya aquesta no queda exposada. En el procés d'inici de sessió, la contrasenya introduïda es torna a processar i es compara amb el hash emmagatzemat.

Un cop autenticat, l'usuari rep l'accés mitjançant JSON Web Tokens (JWT). Un JWT [19] és un token signat pel servidor que conté informació sobre l'usuari i la seva validesa; com que està signat, el servidor pot verificar-ne l'autenticitat sense necessitat de consultar la base de dades a cada petició. S'ha optat per un esquema estàndard de dos tokens, un de vida curta que acompanya cada petició a un endpoint protegit i un de llarga duració,

que serveix únicament per obtenir un nou access token quan aquest caduca, per tant de no obligar l'usuari a tornar a introduir les credencials.

Això ens permet que l'access token, que viatja constantment, tingui una exposició limitada, mentre que el refresh token que és més sensible, s'utilitza poques vegades.

9.2 La barra de navegació i les organitzacions

Un cop iniciada la sessió l'usuari entra a l'aplicació pròpiament dita. Totes les pantalles internes comparteixen una mateixa estructura: a dalt hi ha una barra de navegació fixa, i a sota el contingut de cada pantalla.

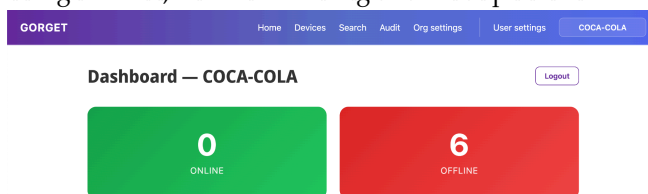


Figura 2: barra de navegació amb el selector d'organització.

La barra conté els enllaços a les seccions principals (dispositius, cerca, auditoria, configuració) i a la dreta, el selector d'organització.

Les organitzacions són les bases del model multi-tenant del sistema, cada organització agrupa els seus propis usuaris i recursos de manera aïllada. Com s'ha descrit al model de dades, la relació entre usuaris i organitzacions és de molts a molts, amb un rol associat a cada vincle. Quan un usuari crea una organització, queda vinculat com a propietari i a partir d'aquell moment pot gestionar-ne els recursos.

En seleccionar una organització l'usuari arriba al dashboard de l'organització. Aquest ofereix una visió de conjunt de l'estat de l'organització activa amb dos comptadors: el nombre dispositius que estan en línia i els que hi són fora de línia, com podem veure a la figura 2. Aquests comptadors no s'emmagatzemen enlloc, sinó que es calculen sobre la marxa a partir de l'estat dels dispositius de l'organització, de manera que sempre reflecteixen la situació actual.

9.3 Llistat de dispositius

La pantalla central de Gorget és l'inventari de dispositius. És on l'usuari veu d'un cop d'ull tots els dispositius que té a l'organització activa: el nom, l'adreça IP, el fabricant, el model, el grup a què pertanyen i el seu estat actual. Des d'aquí també pot afegir dispositius nous o crear grups per organitzar-los. Aquest flux es pot observar en les figures 3 i 4.

GORGET					
Home Devices Search Audit Org settings User settings COCA-COLA					
Devices					
+ New group + New device					
NAME	IP	VENDOR	MODEL	GROUP	STATUS
REAL	79.155.87.18443	fortinet	F40	Planta 3	offline
Despacho Azul	123.23.43.22	fortinet	F-30	Planta 2	offline
IT	123.231.123.12	fortinet	40F	Planta 3	offline
Departamento Ventas	123.42.64.234	fortinet	30F	Edificio Sant Cugat	offline
Despacho Naranja	23.234.23.123	fortinet	30F	Planta 2	offline
Principal	11.22.33.44	fortinet	40F	Planta 2	offline

Figura 3: Taula de dispositius.

GORGET					
Home Devices Search Audit Org settings User settings COCA-COLA					
Devices					
+ New group + New device					
NAME	IP	VENDOR	MODEL	GROUP	STATUS
REAL	79.155.87.18443	fortinet	F40	Planta 3	offline
Despacho Azul	123.23.43.22	fortinet	F-30	Planta 2	offline
IT	123.231.123.12	fortinet	40F	Planta 3	offline
Departamento Ventas	123.42.64.234	fortinet	30F	Edificio Sant Cugat	offline
Despacho Naranja	23.234.23.123	fortinet	30F	Planta 2	offline
Principal	11.22.33.44	fortinet	40F	Planta 2	offline

Add device

Name (e.g. FGT-Madrid-01)

IP address

Token

Model (optional)

No group

Create device

Figura 4: Formulari d'alta de dispositiu.

9.4 Detall de dispositius

En clicar qualsevol dispositiu de la llista s'obre la seva vista de detall. Aquí es concentra tota la informació d'un firewall concret, organitzada en pestanyes: resum, interfícies, VPN, gràfics, registres i accions. En la figura 5 podem observar totes les dades mostrades en el resum del dispositiu.

GORGET					
Home Devices Search Audit Org settings User settings COCA-COLA					
← Back to devices					
REAL					
fortinet - 79.155.122.185:8443 - online					
Summary Interfaces VPN / VPN Users Graphs Logs / Security Actions					
Name	REAL				
IP address	79.155.122.185:8443				
Vendor	fortinet				
Model	F40				
Status	online				

Figura 5: Vista de detall, pestanya de resum.

De les pestanyes previstes en aquesta versió se han desenvolupat dues. La pestanya de resum mostra les dades bàsiques del dispositiu: nom, adreça IP, fabricant, model i estat, en una sola vista neta. La pestanya de gràfics presenta l'evolució de les mètriques del dispositiu al llarg del temps: ús de CPU, memòria i sessions actives. Les pestanyes restants: interfícies, VPN, registres i accions queden reservades per a properes iteracions i, de moment, queden com a esquelets. Els gràfics es dibuixen a partir de les dades històriques que el sistema recull de cada dispositiu. En la figura 6 es poden veure com es mostren.

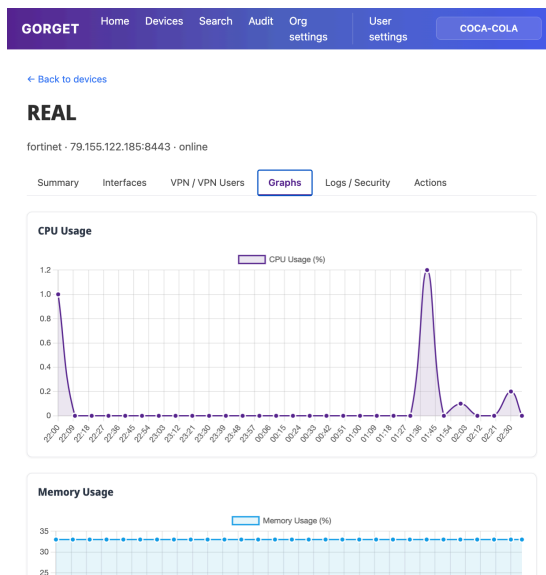


Figura 6: Pestanya de gràfics.

9.5 Monitorització amb polling

Fins ara hem vist què mostra el sistema però no d'on surten les dades pels històrics d'informació dels dispositius, això ho fem amb la monitorització.

La monitorització és la eina que ens permet conèixer en un moment donat, les dades que demanem als dispositius com per exemple el seu estat i rendiment. A diferència de la resta de *features* que reaccionen a accions de l'usuari, la monitorització ha de funcionar de manera autònoma i continua sense que ningú la desencadeni manualment. Això ho aconseguim mitjançant el polling.

El polling consisteix en fer peticions i emmagatzemar les seves respostes periòdicament. El nostre sistema, cada cert temps, recorre la llista sencera de dispositius registrats i demana el seu estat per posteriorment guardar les dades a la base de dades. Aquest procés s'executa mitjançant un scheduler de tasques [20] configurat per llançar-se cada minut.

El polling seqüencial té el problema que fins que no rebem resposta d'un dispositiu no executem la petició del següent. Això provoca que si tenim 10 dispositius amb un temps de transacció de 10 segons el polling triga 100 segons, més del que triga a executar-se la següent ronda. Per evitar-ho s'ha implementat polling concurrent: totes les consultes d'una mateixa ronda es llancen de manera concurrent i s'espera que totes acabin abans de continuar. D'aquesta manera el temps d'una ronda no és la suma dels temps de cada dispositiu, sinó el del més lent.

9.5.1 El client: interfície i adaptadors

Per consultar un dispositiu cal parlar amb la seva API, però no volem que la lògica del polling depengui dels detalls de cada fabricant. Per això s'ha definit una interfície anomenada Client a la capa de domini, que declara les operacions que tot client de monitorització té: `get_status`, per saber si el dispositiu està en línia, i `get_metrics`, per obtenir-ne les dades de rendiment.

D'aquesta interfície en surten dues implementacions a la capa d'infraestructura. La primera és FortiGateClient, el client real que fa les peticions HTTP a l'API del FortiGate. La segona és FortiGateMockClient, un client que retorna dades simulades sense connectar-se a cap dispositiu. Aquest disseny segueix el patró adaptador: el polling treballa sempre contra la interfície Client, sense saber quina implementació concreta hi ha al darrere.

Gràcies a això canviar entre el client real i el simulat és tan senzill com modificar la línia on s'injecta la dependència. El client simulat ha estat molt útil durant el desenvolupament, ja que ha permès provar tot el sistema de monitorització abans de disposar d'un dispositiu físic.

9.5.2 Sèries temporals

Les dades obtingudes de la monitorització dels dispositius són de caràcter diferent a la resta, ja que no són un estat que s'actualitza, sinó que són dades fixes associades a un moment en el temps. Aquest tipus de dades s'anomenen sèries temporals.

En aquest tipus de base de dades cada punt es desa amb un instant de temps, unes etiquetes (tags) que identifiquen a quin dispositiu pertany i uns camps (fields) amb els valors mesurats. Aquesta estructura permet consultar de manera eficient totes les mètriques d'un dispositiu dins d'un rang de temps determinat.

La base de dades tradicional utilitzada pels altres models, PostgreSQL, no és una bona solució per l'eficient manipulació i emmagatzematge d'aquest tipus de dades, per això s'ha optat per una solució especialitzada en aquestes sèries temporals. Per tant, com a base de dades de sèries temporals s'ha escollit InfluxDB. A diferència de PostgreSQL, està dissenyada per ingerir grans volums de dades amb marca de temps i consultar-les per rangs.

Un dels reptes de les sèries temporals és que el volum de dades creix sense parar. Si es vol mostrar un gràfic de les últimes 24 hores i es generen dades cada minut, tindriem més de mil punts per dispositiu, massa per representar de manera útil. Per resoldre-ho s'aplica un downsampling en el moment de la consulta: en lloc de retornar tots els punts, s'agrupen en finestres de temps (per exemple, de 10 minuts) i se'n calcula la mitjana.

10 CONNEXIÓ A DISPOSITIUS

Fins ara el sistema s'havia provat amb el client simulat que retorna dades inventades. El següent pas ha estat connectar el sistema a un dispositiu físic real: un FortiGate, el tallafocs que el sistema està dissenyat per monitoritzar. En la figura 7 podeu observar el dispositiu FortiGate 40F fet servir durant el desenvolupament del projecte.



Figura 7: Dispositiu FortiGate 40F.

10.1 L'API REST de FortiGate

Els dispositius FortiGate exposen una API REST [21] que permet consultar el seu estat i les seves mètriques de manera remota. El sistema en fa servir dos endpoints: un per comprovar l'estat general del dispositiu i un altre per obtenir-ne les mètriques de rendiment (CPU, memòria, disc i sessions).

L'accés a aquesta API està protegit. Per autenticar-se cal generar al dispositiu un token d'API, que s'envia a cada petició dins la capçalera d'autorització. D'aquesta manera el sistema demostra que té permís per consultar el dispositiu sense necessitat d'usuari i contrasenya.

10.2 El repte de la xarxa: accés des del núvol

El sistema està pensat per executar-se al núvol en un servidor accessible des d'internet. Els dispositius a monitoritzar, en canvi, es troben dins de la xarxa local del client darrere del seu router. Això planteja un problema fonamental: un servidor extern no pot accedir directament a un dispositiu d'una xarxa local privada, ja que aquestes xarxes no són visibles des d'internet.

Existeixen diverses maneres d'afrontar aquest problema. La solució més robusta seria un agent de polling instal·lat a la xarxa del client que fes el polling localment i enviés els resultats al núvol. Aquesta opció però, queda fora de l'abast d'aquest projecte i es proposa com a treball futur. Per a la validació amb un dispositiu real s'ha optat per una solució més directa: exposar el dispositiu a internet de manera controlada.

Perquè el servidor del núvol pugui arribar al FortiGate, cal fer que aquest sigui accessible des d'internet. Això s'aconsegueix amb una redirecció de ports (port forwarding) al router de la xarxa local: es configura el router perquè, quan rebí una connexió per un port determinat, la reenvii cap al dispositiu intern corresponent. Exposar la interfície de gestió d'un dispositiu de firewall és un risc de seguretat. Per reduir el risc, s'ha implementat una restricció d'accés de *trusted hosts* a només l'IP pública del nostre servidor.

11 TESTING

Per garantir que el sistema funciona correctament i que els canvis futurs no introdueixin errors, s'ha desenvolupat una bateria de proves automàtiques. Aquesta secció descriu què s'ha decidit provar i com s'ha estructurat aquesta verificació.

11.1 Estratègia de testing

S'ha decidit concentrar les proves en els casos d'ús, és a dir, la capa d'aplicació, ja que és on resideix el flux que crida la lògica de negoci del sistema. És aquí on es prenen les decisions importants (com marcar un dispositiu com a fora de línia o assignar el rol de propietari al crear una organització), i per tant és on un error tindria més impacte.

Cada prova segueix una mateixa estructura, coneguda com a Given-When-Then: primer es prepara l'escenari inicial (Given), després s'executa l'acció que es vol provar (When) i finalment es comprova que el resultat és l'esperat (Then).

S'han escrit proves tant per al camí correcte (per exemple, crear un dispositiu i comprovar que es desa amb les dades correctes) com per als casos d'error (per exemple, demanar un dispositiu que no existeix i comprovar que es llança l'excepció corresponent).

Degut al limitat abast que proporciona el projecte s'ha decidit restringir el testing a les *features* més claus i sensibles de patir modificacions que poden introduir errors: dispositius i polling.

11.2 Arquitectura de testing

El principal repte de provar un cas d'ús és que aquest depèn d'un repositori que accedeix a la base de dades. Si cada prova hagués de connectar-se a una base de dades real les proves serien lentes, fràgils i dependrien d'una infraestructura externa. A més, FastAPI, contrari a alternatives com Django, no proporciona de manera nativa bases de dades de test.

Aquí és on l'arquitectura hexagonal demostra el seu valor. Com que els casos d'ús reben les seves dependències pel constructor i treballen contra interfícies, en lloc del repositori real es pot injectar un repositori fals (fake) que emmagatzema les dades en memòria. El cas d'ús no nota cap diferència ja que el fake ofereix els mateixos mètodes que el repositori real. D'aquesta manera les proves verifiquen la lògica de manera aïllada, sense tocar la base de dades, i s'executen en mil·lèsimes de segon.

Tots els tests passen correctament i ens han donat un feedback important al saber que el funcionament del sistema és el correcte, a més de la confiança de que en futures integracions d'altres funcions, sabrem si aquestes trenquen el comportament existent.

12 DEPLOYMENT

Un cop el sistema funciona en l'entorn local de desenvolupament, calia posar-lo en producció i fer-lo accessible des d'internet. Aquesta secció descriu com s'ha desplegat tant el servidor com el client en un servidor real.

12.1 Configuració del vps

El sistema s'allotja en un servidor virtual privat (VPS) amb sistema operatiu Linux. En aquest servidor hi

conviu totes les peces del sistema: la base de dades PostgreSQL, la base de dades de sèries temporals InfluxDB i l'aplicació del servidor.

El fet que tot s'executi dins el mateix servidor té un avantatge directe: no hi ha sortida de les dades per comunicar-se les diferents parts del sistema. Tot queda dins del servidor i els riscos de seguretat són molt menors.

L'aplicació del servidor s'executa amb uvicorn [22], el servidor que fa funcionar FastAPI. Ara bé, no hi ha prou amb engegar-lo manualment: si el procés s'atura o el servidor es reinicia, l'aplicació deixaria de funcionar. Per resoldre-ho, l'aplicació es gestiona com un servei de systemd, el sistema d'inici de Linux. Això aporta dues garanties: el servei s'engega automàticament quan arrenca el servidor i es reinicia sol si es produeix un error.

Per tal de exposar l'aplicació al internet s'ha fet servir nginx, l'única peça que rep dades i peticions directament des de fora.

12.2 Domini i HTTPS

Perquè el sistema fos accessible des del exterior a través del seu domini i de manera segura es va associar un domini al servidor, *gorget.net* i es va configurar el xifratge HTTPS.

El certificat HTTPS s'obté de manera automàtica i gratuïta mitjançant Let's Encrypt, una autoritat de certificació que permet emetre i renovar certificats sense cost. nginx s'encarrega de redirigir tota petició no xifrada (HTTP) cap a la versió segura (HTTPS), de manera que tota la comunicació entre l'usuari i el sistema viatja xifrada.

12.3 Build del client

El client web desenvolupat amb SvelteKit primer passa per un procés de compilació (build), que el transforma en fitxers estàtics optimitzats (HTML, CSS i JavaScript). Aquests fitxers són els que serveix nginx.

13 RENDIMENT I ESCALABILITAT

Un cop el sistema funciona en producció és pertinent analitzar-ne el comportament sota càrrega: quina latència experimenta l'usuari, quants usuaris i dispositius podria suportar la infraestructura actual i com creixen les necessitats d'emmagatzematge amb el temps.

13.1 Latència de les peticions

La latència és el temps que triga una petició a anar des del navegador de l'usuari fins al servidor i tornar. S'ha mesurat des d'una connexió a Barcelona, Espanya cap al VPS en Nuremberg, Alemanya.

TAULA 2: Latències

Mètrica	Valor mesurat
Connexió (ping al VPS)	72 ms
Resposta d'una petició	155 ms

Processament intern del servidor	1,7 ms
----------------------------------	--------

Veiem a la taula 2 com el servidor triga a processar la petició en 1,7 ms, per tant el coll d'ampolla major està en la transacció en la xarxa.

13.2 Capacitat del servidor

L'aplicació del servidor consumeix aproximadament 84 MB de memòria sobre els 4 GB de RAM del VPS. Això significa que l'aplicació ocupa poc més del 2% i hi cabria desenes de vegades. Per tant la memòria no és el factor limitant.

Gràcies al polling concurrent, el temps d'una ronda no creix amb el nombre de dispositius (és el del més lent, no la suma), de manera que el sistema pot gestionar un nombre molt elevat de dispositius abans no apareguin problemes. El límit pràctic vindria donat pel nombre de connexions simultànies que el servidor pot mantenir obertes i pel processament de cada resposta, que com hem vist és mínim (1,7 ms).

13.3 Emmagatzematge de mètriques

Cada dispositiu genera un punt de dades per minut a InfluxDB, és a dir, 1.440 punts al dia. Cada punt conté quatre mètriques (CPU, memòria, disc i sessions), una etiqueta amb l'identificador del dispositiu i una marca de temps.

InfluxDB comprimeix de manera molt eficient les sèries temporals, de manera que cada punt, un cop comprimit, pesa uns pocs bytes. Per estimar l'espai necessari s'ha mesurat l'ocupació real en disc de la base de dades InfluxDB en l'entorn de producció. Amb 891 lectures emmagatzemades, la base de dades ocupava 540 KB en disc, cosa que dona una mitjana d'uns 620 bytes per lectura.

Cal matisar aquesta xifra: InfluxDB té un cost fix d'estructura que no depèn del nombre de punts. Amb una base de dades tan petita aquest cost domina i infla el cost per punt; a gran escala s'amortitza i l'espai real per lectura baixa. Per tant 620 bytes per lectura és el pitjor cas, no el cost a gran escala. En la taula 3 podem veure la mida de les dades en diferents franges temporals.

TAULA 3: Dades temporals per temps

Període	Lectures per dispositiu	Mida (pitjor cas)
1 dia	1 440	~0,9 MB
1 mes	43 200	~27 MB
1 any	525 600	~326 MB

13.4 Optimització: downsampling progressiu

A mesura que les dades envelleixen deixa de tenir sentit conservar-les amb la màxima resolució. Ningú necessita saber l'ús de CPU minut a minut de fa tres mesos, hi ha

prou amb una mitjana per hora.

Per això es proposa una política de retenció amb downsampling progressiu: les dades recents es conserven a resolució completa (un punt per minut), però a partir d'un cert temps (per exemple, 24 hores) es substitueixen per una mitjana per hora, i les més antigues, per una mitjana diària, com podem veure a la taula 4. Això redueix dràsticament l'espai ocupat sense perdre la informació rellevant a llarg termini.

TAULA 4: Downsampling

Antiguitat de les dades	Resolució	Punts per dispositiu per dia
0-24 h	1 minut	1440
1-30 dies	1 hora	24
> 30 dies	1 día	1

14 CONCLUSIONS

L'objectiu d'aquest treball ha estat construir Gorget, una plataforma online per a la monitorització i gestió de dispositius de firewalls de xarxa FortiGate. Ara que el projecte ha estat completat, podem afirmar que el nucli del sistema ha estat assolit exitosament. Hem aconseguit una aplicació funcional desplegada al internet, on podem afegir dispositius i monitoritzar-los, encara que de manera limitada.

Més enllà del producte concret el veritable valor del projecte ha estat l'experiència de desenvolupar un producte des de la seva concepció com a idea fins tenir-lo materialitzat i desplegat. El procés ha recorregut totes les etapes del desenvolupament del software, des de la visió de producte, anàlisi i definició de requisits, modelatge de l'arquitectura i decidir tecnologies, desenvolupar-lo organitzadament, fer infinites proves i finalment, desplegar-lo i donar-li una identitat i marca.

A nivell d'aplicació aquest projecte permet extreure'n una conclusió a nivell de producte: és viable construir una eina de gestió i monitorització de firewalls accessible i de baix cost. El sistema demostra que funcionalitats que tradicionalment requereixen programari empresarial complex i car (com FortiManager) es poden oferir a través d'una plataforma web senzilla, accessible des de qualsevol lloc i sense necessitat d'instal·lar res al costat de l'usuari. La infraestructura necessària per fer-ho funcionar és modesta, cosa que confirma que el cost d'operació pot ser baix.

Cal ser realista sobre la maduresa actual. El que s'ha construït és una base sòlida i funcional, no un producte llest per al mercat. Per arribar-hi caldria resoldre el repte de la connexió (l'agent de polling en xarxes locals), ampliar les funcionalitats i afegir la robustesa que un entorn de producció real exigeix, tal com es detalla a continuació.

15 MILLORES FUTURES

Tot i haver assolit el nucli del sistema, el projecte té un marge ampli de creixement. Algunes funcionalitats van quedar fora de l'abast i es proposen com a continuació natural:

- Agent polling local. L'evolució més rellevant a nivell arquitectònic. En lloc d'exposar el dispositiu a internet, un agent instal·lat a la xarxa del client faria el polling localment i enviaria els resultats al núvol mitjançant una connexió sortint. És l'enfocament que utilitzen les solucions professionals, i resoldria de manera neta els reptes de xarxa i seguretat descrits a la secció 10.

- Més funcionalitats del FortiGate. L'API del dispositiu ofereix molta més informació de la que actualment s'aprofita: dades del sistema, estat de les interfícies, túnels VPN, registres de seguretat o còpies de seguretat de la configuració. Incorporar-les faria el producte molt més complet.

- Funcions clau de producte: Reports, Billing, Mapes i dades geogràfiques i Cerca global.

- Optimització de les dades. Implementar la política de retenció amb downsampling progressiu proposada a la secció 13, per garantir l'escalabilitat de l'emmagatzematge a llarg termini.

AGRAÏMENTS

En primer lloc vull agrair al meu tutor del treball de fi de grau, en Jorge Bernal, per tot el seu suport durant el desenvolupament d'aquest projecte. En especial vull agrair la seva sinceritat davant el projecte i per adaptar-se als meus horaris canviants. Ha estat un plaer tenir un tutor tan involucrat, ja que sense aquesta participació activa per part seva, el projecte no hauria arribat a aquest punt de maduresa.

També agrair a tot el personal de la Universitat Autònoma de Barcelona per acompanyar-me en aquest procés per convertir-me en enginyer, en especial al coordinador del grau, Jordi Pons, per donar-me suport tots aquests anys, tant en els bons moments com els més delicats.

Vull agrair a la meva família, per donar-me tot el suport perquè jo em pugui concentrar al complet en els estudis i carrera laboral, sense mai demanar res a canvi. També agrair al meu germà, Aitor Osuna, qui ha fet possible aquest projecte, ja que la idea primera va ser seva. Quan me la va explicar vaig pensar, "quin bon tfg seria", i quin bon tfg ha estat. Moltes gràcies, segurament sense ell ara mateix no seria Enginyer.

Agrair també al meu tiet, qui va emmalaltir i morir durant el desenvolupament del projecte, el qual sempre estava orgullós dels seus nebots, futurs enginyers.

Per últim agrair al meu avi, ell va ser qui va despertar en mi la curiositat per fer realitat les meves idees, així com em va fer veure que no hi ha res més satisfactori que construir, sigui el que sigui, per fer un món una mica millor. Moltes gràcies i tant de bo et faci il·lusió la persona en qui m'estic convertint.

BIBLIOGRAFIA

- [1] Hetzner Online GmbH, "Hetzner Cloud." [En línia]. Disponible: <https://www.hetzner.com/cloud>. [Accés: juny 2026].
- [2] S. Ramírez, "FastAPI documentation." [En línia]. Disponible: <https://fastapi.tiangolo.com/>. [Accés: juny 2026].
- [3] Svelte, "SvelteKit documentation." [En línia]. Disponible: <https://svelte.dev/docs/kit/>. [Accés: juny 2026].
- [4] The PostgreSQL Global Development Group, "PostgreSQL documentation." [En línia]. Disponible: <https://www.postgresql.org/docs/>. [Accés: juny 2026].
- [5] InfluxData, "InfluxDB documentation." [En línia]. Disponible: <https://docs.influxdata.com/influxdb/>. [Accés: juny 2026].
- [6] Software Freedom Conservancy, "Git documentation." [En línia]. Disponible: <https://git-scm.com/doc>. [Accés: juny 2026].
- [7] GitHub, "GitHub documentation." [En línia]. Disponible: <https://docs.github.com/>. [Accés: juny 2026].
- [8] F5, Inc., "nginx documentation." [En línia]. Disponible: <https://nginx.org/en/docs/>. [Accés: juny 2026].
- [9] freedesktop.org, "systemd System and Service Manager." [En línia]. Disponible: <https://www.freedesktop.org/wiki/Software/systemd/>. [Accés: juny 2026].
- [10] Internet Security Research Group, "Let's Encrypt documentation." [En línia]. Disponible: <https://letsencrypt.org/docs/>. [Accés: juny 2026].
- [11] A. Cockburn, "Hexagonal architecture." [En línia]. Disponible: <https://alistair.cockburn.us/hexagonal-architecture/>. [Accés: juny 2026].
- [12] R. C. Martin, "The Clean Architecture," The Clean Code Blog, 2012. [En línia]. Disponible: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. [Accés: juny 2026].
- [13] Pydantic, "Pydantic documentation." [En línia]. Disponible: <https://docs.pydantic.dev/>. [Accés: juny 2026].
- [14] M. Bayer, "SQLAlchemy 2.0 documentation." [En línia]. Disponible: <https://docs.sqlalchemy.org/en/20/>. [Accés: juny 2026].
- [15] Mozilla, "Using the Fetch API," MDN Web Docs. [En línia]. Disponible: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch. [Accés: juny 2026].
- [16] M. Bayer, "Alembic documentation." [En línia]. Disponible: <https://alembic.sqlalchemy.org/>. [Accés: juny 2026].
- [17] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," tesi doctoral, Univ. of California, Irvine, 2000. [En línia]. Disponible: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. [Accés: juny 2026].
- [18] N. Provos i D. Mazières, "A Future-Adaptable Password Scheme," in Proc. USENIX Annual Technical Conference, 1999.
- [19] M. Jones, J. Bradley i N. Sakimura, "JSON Web Token (JWT)," RFC 7519, IETF, 2015. [En línia]. Disponible: <https://datatracker.ietf.org/doc/html/rfc7519>. [Accés: juny 2026].
- [20] A. Grünfeld, "APScheduler documentation." [En línia]. Disponible: <https://apscheduler.readthedocs.io/>. [Accés: juny 2026].
- [21] Fortinet, Inc., "FortiOS REST API," Fortinet Developer Network. [En línia]. Disponible: <https://fdn.fortinet.net/>. [Accés: juny 2026].
- [22] Encode, "Uvicorn documentation." [En línia]. Disponible: <https://www.uvicorn.org/>. [Accés: juny 2026].
- [23] Mermaid, "Class diagrams." [En línia]. Disponible: <https://mermaid.js.org/syntax/classDiagram.html>. [Accés: juny 2026].
- [24] A. Bryksin, "Kotlin Clean Architecture and CQRS," Medium, 2021. [En línia]. Disponible: <https://alexbyksin.medium.com/kotlin-clean-architecture-and-cQRS-74f8f0cacca8>. [Accés: juny 2026].

APÈNDIX

A1. FIGURES EXTRA

Aquí es detallen les figures extra que no han pogut ser incloses al document principal. Els diagrames d'aquesta memòria s'han generat amb Mermaid [23].

Figura 8: Gantt de la planificació.

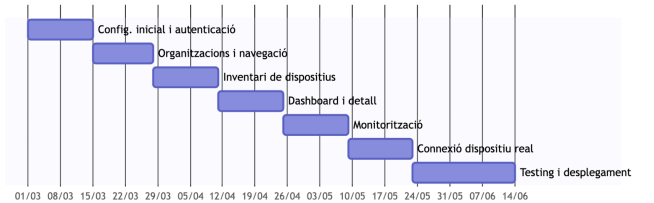


Figura 9: Pantalla de login.

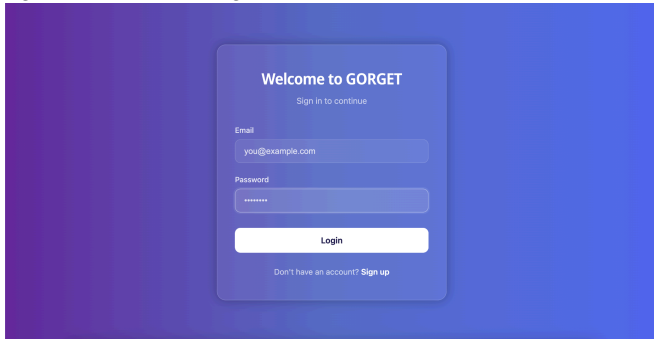


Figura 10: Arquitectura hexagonal del codi

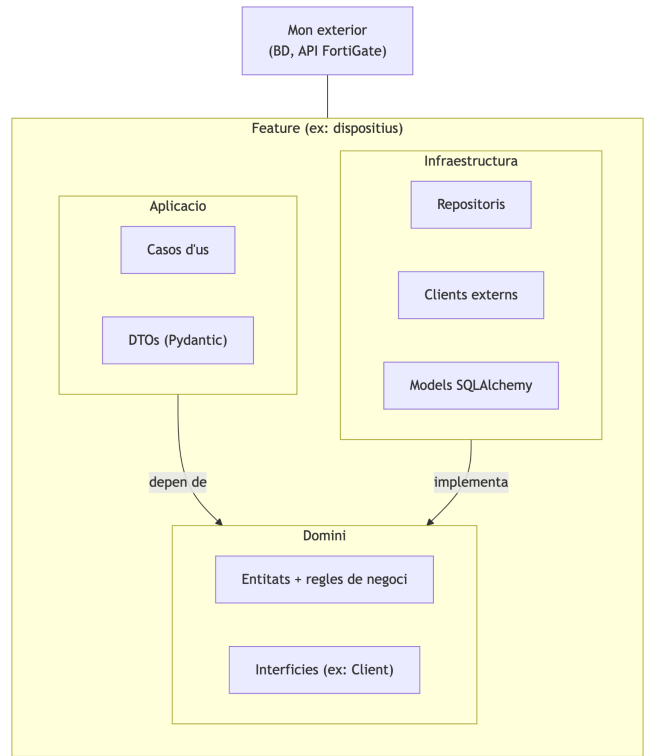


Figura 11: Models de dades

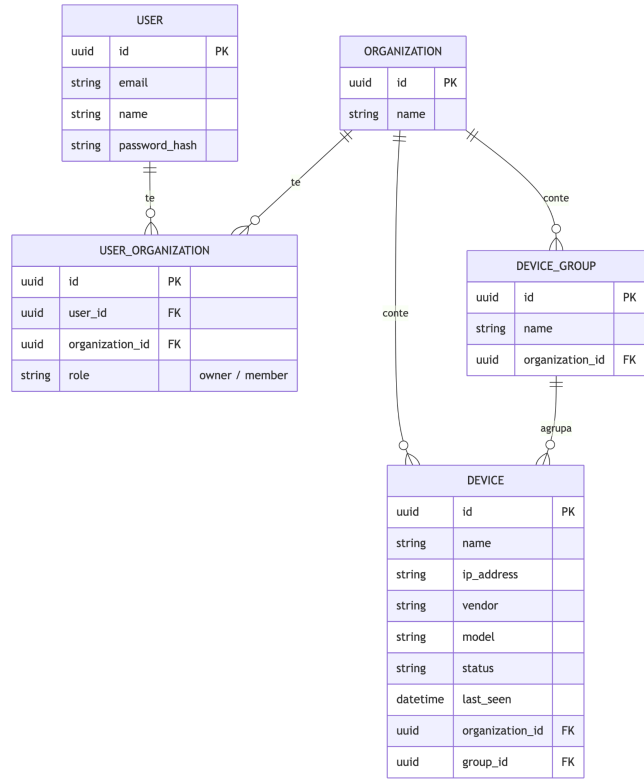


Figura 12: Arquitectura del sistema

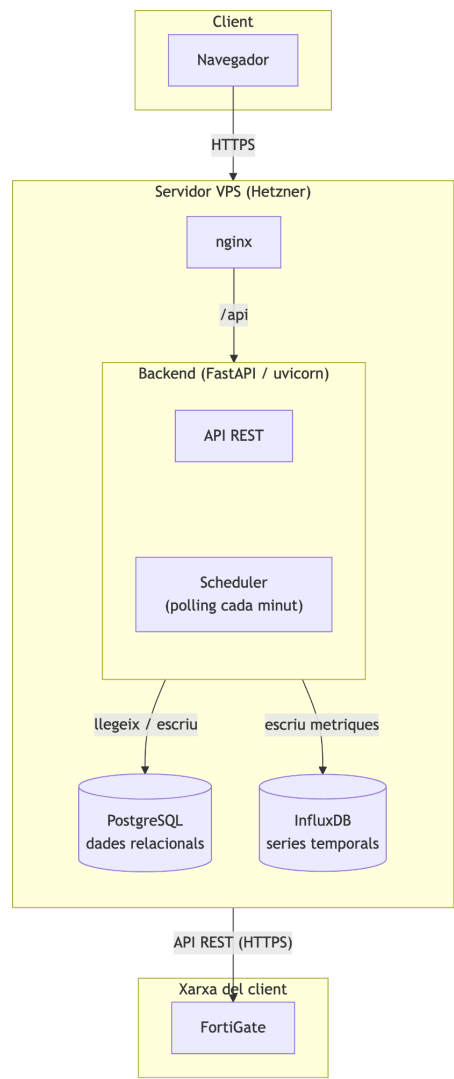


Figura 13: Documentació de la api generada per FastAPI

Gorget API

0.1.0

0.1.0

0.1.0

OpenAPI JSON

Authorize

auth

POST /auth/register Register

POST /auth/login Login

POST /auth/refresh Refresh

organizations

GET /organizations List Orgs

POST /organizations Create

devices

POST /devices Create Device

GET /devices List Devices

POST /devices/groups Create Group

GET /devices/groups List Groups

GET /devices/{device_id} Get Device

monitoring

POST /monitoring/poll Poll

GET /monitoring/metrics/{device_id} Get Metrics